

Il processo in Unix; caratteristiche e rappresentazione

Il processo è l'unità di computazione di Unix.

Si tratta di processi pesanti e con codice rientrante: non hanno cioè dati condivisi, mentre il codice è condivisibile come altri processi.

La struttura dati essenziale è la Process Structure: è il descrittore del processo e contiene tutti i dati che è necessario conoscere quando un processo è swappato: pid, stato del processo, informazioni sullo scheduling, puntatori al padre, al figlio, ...

Lo spazio degli indirizzi virtuali di un processo utente è suddiviso in segmenti di dati e stack.

Ogni processo ha nella sua Process Structure un puntatore ai codici utilizzati ed eventualmente condivisi dai processi (Text Table, contenuta nella Text Structure).

Le Text Structure sono una per ogni processo allocato in memoria e contengono ad esempio il numero dei processi che la condividono e il puntatore al codice.

Le informazioni sul processo che servono solo quando questo è residente in memoria, sono nella User Structure. In essa c'è anche lo spazio per contenere i parametri delle system calls.

I processi in Unix infatti hanno funzionamento dual mode, con diverse potenzialità e diversa visibilità in memoria:

- modo utente: avviene la maggior parte del lavoro ordinario
- modo kernel: modalità con cui vengono eseguite le system calls (es fork, wait, exec,...)

La sincronizzazione tra processi in Unix

La sincronizzazione tra processi permette di imporre vincoli sulle operazioni dei processi interagenti, ad esempio per imporre un particolare ordine cronologico nelle azioni eseguite dai processi interagenti, o per garantire la mutua esclusione dei processi a una risorsa condivisa.

Unix è un s.o. ad ambiente globale, quindi non vi è la possibilità di condividere memoria né variabili.

La sincronizzazione può essere realizzata mediante l'opportuno utilizzo di segnali generati dal terminale, da altri processi, o dal kernel in seguito ad eccezioni hw o a condizioni sw.

Quando un processo riceve un segnale, può comportarsi:

- ignorando il segnale
- eseguendo un'azione di default per quel segnale
- gestendo il segnale con una funzione handler definita dal programmatore

Negli ultimi 2 casi, il processo reagisce in modo asincrono: interrompe la sua esecuzione, esegue l'azione associata (default o handler), e ritorna all'istruzione successiva al segnale.

Generalmente la system call kill è quella utilizzata da un processo per inviare un segnale ad un altro processo attualmente sospeso.

La comunicazione tra processi in Unix

Unix è un modello ad ambiente globale, quindi i processi non possono condividere memoria.

Per ovviare a questo problema si ricorre a condivisione file o all'utilizzo di pipe (per processi della stessa gerarchia), di fifo (per processi sulla stessa macchina), o di socket (per processi in nodi della stessa rete).

La pipe è un canale di comunicazione unidirezionale di tipo molti a molti e di capacità limitata. La comunicazione è di tipo indiretto, cioè senza naming esplicito. La pipe inoltre viene distrutta quando terminano tutti i processi che la usano.

La fifo è un canale di comunicazione unidirezionale, è persistente (rappresentata da un file nel file-system) e ha visibilità potenzialmente globale.

Il file system

Il file system è quella parte del s.o. che fornisce i meccanismi di accesso e memorizzazione dei programmi e dei dati allocati in memoria di massa.

Realizza i concetti astratti di file, direttorio e partizione.

Il file è un insieme di informazioni rappresentato come insieme di record logici (ciò che il processo trasferisce da/verso il file). I record logici sono contenuti all'interno di blocchi, cioè unità di allocazione delle unità sul disco.

Il file system deve stabilire la corrispondenza tra record logici e blocchi. Questi ultimi, possono essere allocati sul disco in modi differenti (alcuni s.o. utilizzano per tipi di file diversi tipi diversi di allocazione):

- allocazione contigua: ogni file è mappato su un insieme di blocchi fisicamente contigui. (problema principale: frammentazione esterna);
- allocazione a lista: i blocchi sono organizzati in una lista concatenata (FAT). (problema principale: costo della ricerca di un blocco)
- allocazione a indice: come l'allocazione a lista, solo che i puntatori ai blocchi utilizzati per l'allocazione dei file sono concentrati in un blocco indice. (problema principale: scarso utilizzo del blocco indice)

Organizzazione fisica del file system di unix

In Unix tutto è file: ne esistono 3 categorie: i file ordinari, i direttori e i dispositivi fisici.

Ad ogni file può essere associato più di un nome simbolico (tramite gli hard link), ma c'è un solo descrittore (i-node) identificato da un i-number.

L'i-node contiene gli attributi del file, tra cui il tipo di file, il proprietario, la data, il numero di link, 13-15 indirizzi di blocchi di dati, etc.

La superficie del file system è partizionata in 4 blocchi:

- boot block: blocco in cui è memorizzato il codice di bootstrap
- super block: fornisce i limiti dei 4 blocchi, e i puntatori alla lista dei blocchi e degli i-node liberi
- data block: l'area per la memorizzazione dei files
- i-list: contiene la lista di tutti gli i-node dei file del file system

L'allocazione dei file è a indice, e non è su blocchi contigui, ma nell'i-node sono contenuti i puntatori ai blocchi di dati ed eventualmente ad altri blocchi indice contenenti indirizzi di blocchi di dati.

I direttori sono visti anch'essi come files, e contengono un insieme di record logici rappresentanti i file del direttorio.

Accesso a file di processi Unix: strutture dati del s.o. e loro gestione

Per realizzare l'accesso a file si usano 2 strutture dati globali

- tabella dei file attivi: contiene un elemento per ogni file aperto nel sistema
- tabella dei file aperti di sistema: un elemento per ogni sessione di accesso al file nel sistema.

Inoltre, ad ogni processo è associata una tabella dei file aperti, in cui ogni elemento è un file aperto dal processo ed è individuato da un file descriptor.

Unix permette ai processi di accedere a file mediante una serie di system call: open, creat, close, write, read, lseek, link, unlink, ...

All'apertura di un file viene inserito un nuovo elemento nella prima posizione libera dei file aperti di processo; viene inserito un nuovo record nella tabella dei file aperti di sistema, e viene copiato l'i-node del file nella tabella dei file attivi.

Scheduling della cpu

Lo scheduling della cpu è l'attività mediante la quale il sistema operativo effettua le scelte tra i processi riguardo l'assegnazione della cpu.

Lo scheduler è quella parte del s.o. che seleziona dalla coda dei processi pronti il processo a cui assegnare la cpu.

Nei sistemi time-sharing, allo scadere di un quanto di tempo, il s.o. decide a quale processo assegnare la cpu ed effettuare il cambio di contesto (realizzato dal dispatcher).

La coda dei processi pronti contiene i descrittori (PCB) dei processi pronti, e viene gestita mediante algoritmi di scheduling:

- senza prelazione: la cpu rimane allocata al processo running finchè questo non si sospende volontariamente o termina

- con prelazione: il s.o. può sottrarre la cpu al processo running per assegnarla a un altro.

First Come First Served: (algoritmo senza prelazione) la coda viene gestita in modo fifo. Non è possibile influire sull'ordine dei processi: il tempo di attesa è alto se ci sono molti processi con cpu-burst all'inizio, e si può verificare l'effetto convoglio, per cui la cpu viene poco utilizzata.

Shortest Job First: (tipicamente senza prelazione) per ogni processo nella coda viene stimata la lunghezza del prossimo cpu-burst e viene schedulato il processo con il cpu-burst minore.

Scheduling con priorità: (con/senza prelazione) ad ogni processo viene data una priorità (costante o variabile) e processi con uguale priorità vengono trattati con modo fcfs. C'è il problema che alcuni processi a priorità bassa vengano lasciati sempre in coda, quindi si utilizza un meccanismo di aging per cui la priorità aumenta per i processi da più tempo in attesa e viceversa.

Round Robin: (prelazione periodica) ai processi viene allocata la cpu ciclicamente e per un intervallo di tempo costante.

Tecniche di allocazione della memoria centrale

Ai fini della multiprogrammazione si rende necessario poter allocare in memoria più processi contemporaneamente. La memoria centrale è un vettore di celle, ognuna univocamente individuata da un indirizzo (fisico).

Esistono anche indirizzi simbolici, generati dalla cpu, a cui corrispondono gli indirizzi fisici. La corrispondenza tra indirizzo fisico e logico è detta binding e può essere:

- statico: ad ogni indirizzo logico corrisponde un indirizzo fisico settato in fase di compilazione o caricamento

- dinamico: gli indirizzi logici non corrispondono agli indirizzi fisici e sono settati run time

In quest'ultimo modo è possibile spostare processi in memoria e swapparne altri (attraverso il memory management unit).

Esistono varie tecniche di allocazione di dati e codice in memoria centrale:

- Allocazione contigua a partizione singola e a partizioni multiple

- Allocazione non contigua: paginazione o segmentazione

Allocazione contigua a partizione singola: la memoria non è partizionata e può essere allocato solo un processo alla volta. Non c'è multiprogrammazione

Allocazione contigua a partizioni multiple: ad ogni processo caricato viene associata un'area di memoria distinta. Le partizioni in cui allocare i processi sono fisse o variabili.

-fisse: quando un processo viene schedulato, il so cerca una partizione libera di dimensione abbastanza grande per accoglierlo (frammentazione interna).

-variabili: ogni partizione è allocata dinamicamente: si elimina la frammentazione interna e la multiprogrammazione è variabile, ma bisogna scegliere l'area in cui allocare il processo, e subentra il problema della frammentazione esterna

Paginazione: viene partizionato lo spazio fisico della memoria in pagine di dimensione costante e limitata su cui mappare porzioni dei processi da allocare. In questo modo non c'è più frammentazione esterna (pagine logiche contigue possono essere mappate su pagine fisiche non contigue), la frammentazione interna è limitata (grazie alla dimensione fissa delle pagine).

Segmentazione: si basa sul partizionamento dello spazio logico degli indirizzi in segmenti, caratterizzati da nome e lunghezza. Ogni segmento è allocato in memoria in modo contiguo, ed è riferito al so tramite un intero. Il problema principale in questo caso è la frammentazione esterna; per ovviare a ciò sarebbe necessario allocare i processi con tecniche di tipo best fit, worst fit, ...

Segmentazione paginata: una combinazione tra segmentazione e paginazione: lo spazio logico è diviso in segmenti a loro volta divisi in pagine. Si può così eliminare la frammentazione esterna, ed è possibile caricare in memoria solo le pagine necessarie.

Memoria virtuale

La dimensione della memoria può rappresentare un vincolo riguardo la dimensione dei processi e il grado di multiprogrammazione. La tecnica della memoria virtuale svincola il grado di multiprogrammazione dalla dimensione effettiva della memoria; in questo modo è possibile l'esecuzione di processi non completamente allocati in memoria.

Così si ha uno spazio virtuale illimitato, una maggiore astrazione e un costo minore per quanti riguarda caricamento e swapping di un processo.

Di solito la memoria virtuale è realizzata mediante paginazione su richiesta: il pager (il modulo del so che realizza i trasferimenti da/verso la memoria) trasferisce una pagina in memoria solo se ritenuta necessaria.

In generale una pagina dello spazio logico di un processo può essere in memoria centrale o secondaria: per rappresentare queste 2 opportunità, ogni pagina di un processo ha un bit di validità: è 1 se la pagina è in memoria centrale, è 0 se la pagina è invalida, cioè se non esiste lo spazio logico del processo, o la pagina non è in memoria centrale. In questo caso scatta il meccanismo di paginazione su richiesta, tramite un interrupt (page fault) per cui la pagina viene copiata su un frame libero e il processo riprende l'esecuzione dal punto in cui era stato interrotto, ignaro che la pagina non fosse al momento disponibile in memoria centrale.

In seguito a un page fault è possibile che non ci siano frames liberi: si utilizza il metodo di sovrallocazione per cui viene individuata una vittima che sarà salvata sul disco per far posto alla nuova pagina. Per compiere questa sostituzione bisogna fare 2 trasferimenti da/verso il disco, ma se la vittima non è stata modificata, può essere evitato un trasferimento. A tal fine si introduce un bit di modifica (dirty bit).

La vittima può essere individuata secondo vari algoritmi:

-FIFO: viene sostituita la pagina da più tempo in memoria indipendentemente dal suo uso

-Last Frequently Used: sostituita la pagina usata meno frequentemente. In questo modo è necessario un contatore degli accessi ad ogni pagina.

-Last Recently Used (+ usata): sostituita la pagina usata meno recentemente. È necessario registrare la sequenza degli accessi (time stamping), che provoca un overhead dovuto all'aggiornamento.